# Performance Evaluation of the Quadrics Interconnection Network[*]

Fabrizio Petrini[⋆], Salvador Coll[⋆†], Eitan Frachtenberg[⋆] and Adolfy Hoisie[⋆]

[⋆] CCS-3 Modeling, Algorithms, & Informatics
Computer & Computational Sciences Division
Los Alamos National Laboratory
[†] Electronic Engineering Department
Technical University of Valencia
{fabrizio,scoll,eitanf,hoisie}@lanl.gov

## Abstract

*In this paper we present an in-depth description of the Quadrics interconnection network (QsNET) and an experimental performance evaluation on a 64-node AlphaServer cluster. We explore several performance dimensions and scaling properties of the network by using a collection of benchmarks, based on different traffic patterns. Experiments with permutation patterns and uniform traffic are conducted to illustrate the basic characteristics of the interconnect under conditions commonly created by parallel scientific applications. Moreover, the behavior of the QsNET under I/O traffic, and the influence of the placement of the I/O servers are analyzed. The effects of using dedicated I/O nodes or shared I/O nodes are also exposed. In addition, we evaluate how background I/O traffic interferes with other parallel applications running concurrently. The experimental results indicate that the QsNET provides excellent performance in most cases, with excellent contention resolution mechanisms. Some important guidelines for applications and I/O servers mapping on large-scale clusters are also given.*

**Keywords**: Interconnection Networks, Performance Evaluation, User-level Communication, Operating System Bypass.

## 1 Introduction

Some interconnect technologies used in high-performance computers include Gigabit Ethernet [22], Giganet [25], SCI [8], Myrinet [2] and GSN (HIPPI 6400) [24][1]. Each one provides a different level of programmability, raw performance and integration with the operating-system.

InfiniBand [1] has been recently proposed as a standard for communication between processing nodes and I/O devices as well as for interprocessor communication, offering an integrated view of computing, networking and storage technologies. The InfiniBand architecture is based on a switch interconnect technology with high speed point-to-point links and offers support for Quality of Service (QoS), fault-tolerance, remote direct memory access, etc.

The Quadrics interconnection network (QsNET) provides a number of innovative design issues, some of which are very similar to those defined by the InfiniBand specification. Some of these salient aspects are the presence of a programmable processor in the network interface that allows the implementation of intelligent communication protocols, fault-tolerance, and remote direct memory access. In addition, the QsNET integrates the local virtual memory into a distributed virtual shared memory. The Quadrics network is currently utilized in some of the largest parallel systems in the world, mainly connecting Compaq Alpha-based servers, but increasingly other compute platforms too.

The performance of interconnection networks and, in particular, switch-based wormhole networks has been extensively analyzed by simulation in the literature [11] [3]. Since performance is strongly influenced by the load, it is very important to evaluate the QsNET under more varied traffic patterns to get a complete view of the network behavior. The patterns considered in this work are representative of real scientific applications in use at Los Alamos National Laboratory. One example of workload analysis is presented in [9] for SAGE (SAIC's Adaptive Grid Eulerian hydrocode), an important ASCI application.

In [13] we analyzed the QsNET performance under specific load conditions to obtain the "peak performance" of the network and a baseline for further studies. In this paper we analyze the behavior of the Quadrics interconnect with a much broader set

[*]The work was supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36

[1]See also http://public.lanl.gov/radiant/hippi.html

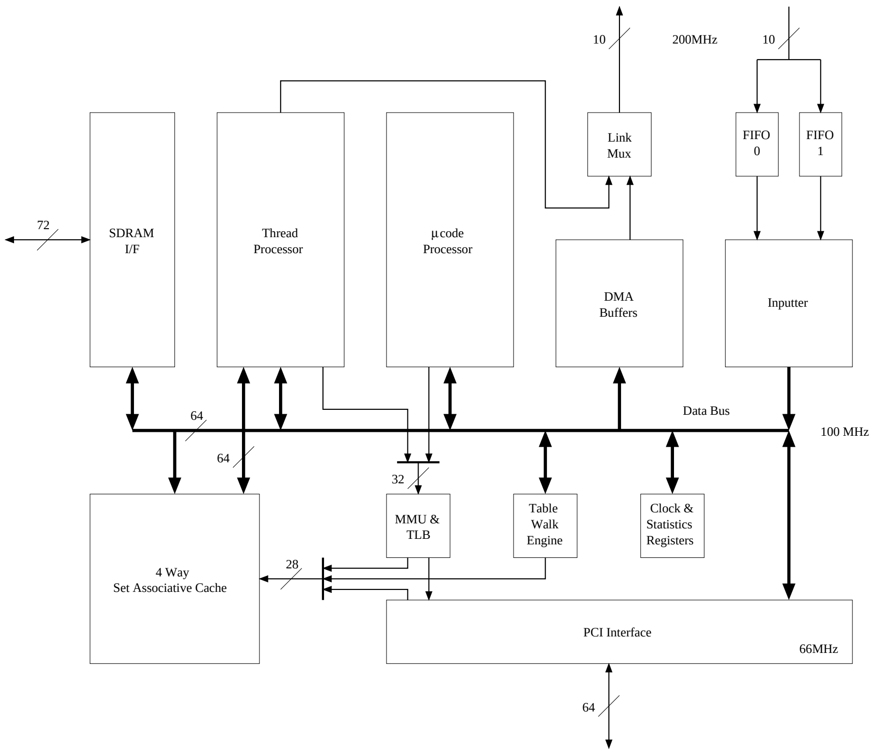


**Figure 1. Elan Functional Units**

of workload conditions. Since the efficient integration of the network with the I/O is a key factor to efficiently exploit the power of high-performance parallel computers, we also address the evaluation of the networking and the I/O integration in the QsNET.

The test bed for the network evaluation was a 64-node QsNET-based AlphaServer ES40 cluster. The performance effects in the network are explained in terms of hardware parameters, flow control and congestion resolution. The paper introduces a number of novel concepts in analyzing network performance, such as the congestion matrix in section 5.4. The results of the analysis in this work have powerful and direct practical implications, for example related to the I/O node mapping and application distribution. The study of the hardware and software primitives used to implement multicast communication and the impact of the network performance on the user applications are outside the scope of the paper and can be found in [14] and [9], respectively.

The rest of this paper is organized as follows. Section 2 gives a comprehensive presentation of the network hardware building blocks. Section 3 discusses the hierarchy of communication libraries. In Section 4 a description of the experimental methodology is given, while Section 5 presents the experimental results and performance analysis. Finally, some concluding remarks are drawn in Section 6.

## 2 The QsNET

The QsNET is based on two building blocks, a programmable network interface called Elan [20] and a low-latency high-bandwidth communication switch called Elite [21]. Elites can be interconnected in a fat-tree topology [10]. The network has several layers of communication libraries which provide trade-offs between performance and ease of use. Other important features are hardware support for collective communication and fault-tolerance.

### 2.1 Elan

The Elan[2] network interface links the high-performance, multi-stage Quadrics network to a processing node containing one or more CPUs. In addition to generating and accepting packets to and from the network, the Elan also provides substantial

[2]This paper refers to the Elan3 version of the Elan. We will use Elan and Elan3 interchangeably throughout the paper.

local processing power to implement high-level message-passing protocols such as MPI. The internal functional structure of the Elan, shown in Figure 1, centers around two primary processing engines: the microcode processor and the thread processor.

The 32-bit microcode processor supports four separate threads of execution, where each thread can independently issue pipelined memory requests to the memory system. Up to eight requests can be outstanding at any given time. The scheduling for the microcode processor is lightweight, enabling a thread to wake up, schedule a new memory access on the result of a previous memory access, and then go back to sleep in as few as two system-clock cycles.

The four microcode threads are described below: (1) *inputter thread:* Handles input transactions from the network. (2) *DMA thread:* Generates DMA packets to be written to the network, prioritizes outstanding DMAs, and time-slices large DMAs so that small DMAs are not adversely blocked. (3) *processor-scheduling thread:* Prioritizes and controls the scheduling and descheduling of the thread processor. (4) *command-processor thread:* Handles operations requested by the host processor at user level.

The thread processor is a 32-bit RISC processor used to aid the implementation of higher-level messaging libraries without explicit intervention from the main CPU. In order to better support such an implementation, the thread processor's instruction set was augmented with extra instructions that construct network packets, manipulate events, efficiently schedule threads, and block-save and restore a thread's state when scheduling.

The Elan contains routing tables that translate every virtual process number into a sequence of tags that determine the network route. Several routing tables can be loaded in order to have different routing strategies. The link logic transmits and receives data from the network and outputs 9 bits and a clock signal on each half of the clock cycle. The flit encoding scheme allows data and command tokens to be interleaved on the link and prevents a corrupted data word being interpreted as a token or a token being interpreted as another token. Each link provides buffer space for two virtual channels with a 128-entry, 16-bit FIFO RAM for flow control.

## 2.2 Elite

The other building block of the QsNET is the Elite switch. The Elite provides the following features: (1) 8 bidirectional links supporting two virtual channels in each direction, (2) an internal $16 \times 8$ full crossbar switch[3], (3) a nominal transmission bandwidth of 400 MB/s on each link direction and a flow through latency of 35 ns, (4) packet error detection and recovery, with routing and data transactions CRC protected, (5) two priority levels combined with an aging mechanism to ensure a fair delivery of packets in the same priority level, (6) hardware support for broadcasts, (7) and adaptive routing.

The Elite switches are interconnected in a quaternary fat-tree topology, which belongs to the more general class of the $k$-ary $n$-trees [17] [16]. A quaternary fat-tree of dimension $n$ is composed of $4^n$ processing nodes and $n * 4^{n-1}$ switches interconnected as a delta network, and can be recursively built by connecting 4 quaternary fat trees of dimension $n-1$. Quaternary fat trees of dimension 1, 2 and 3 are shown in Figure 2.

### 2.2.1 Packet Routing and Flow Control

Each user- and system-level message is chunked in a sequence of packets by the Elan. An Elan packet contains three main components. The packet starts with the (1) routing information, that determines how the packet will reach the destination. This information is followed by (2) one or more transactions consisting of some header information, a remote memory address, the context identifier and a chunk of data, which can be up to 64 bytes in the current implementation. The packet is terminated by (3) an end of packet (EOP) token, as shown in Figure 3.

Transactions fall into two categories: write block transactions and non-write block transactions.

The purpose of a write block transaction is to write a block of data from the source node to the destination node, using the destination address contained in the transaction immediately before the data. A DMA operation is implemented as a sequence of write block transactions, partitioned into one or more packets (a packet normally contains 5 write block transactions of 64 bytes each, for a total of 320 bytes of data payload per packet).

The non-write block transactions implement a family of relatively low level communication and synchronization primitives. For example, non-write block transactions can atomically perform remote test-and-write or fetch-and-add and return the result of the remote operation to the source, and can be used as building blocks for more sophisticated distributed algorithms.

Elite networks are source routed. The routing information is attached to the header before injecting the packet into the network and is composed by a sequence of Elite link tags. As the packet moves inside the network, each Elite removes the first routing tag from the header, and forwards the packet to the next Elite in the route or to the final destination. The routing tag can identify either a single output link or a group of adjacent links.

[3] The crossbar has two input ports for each input link, to accommodate the two virtual channels.

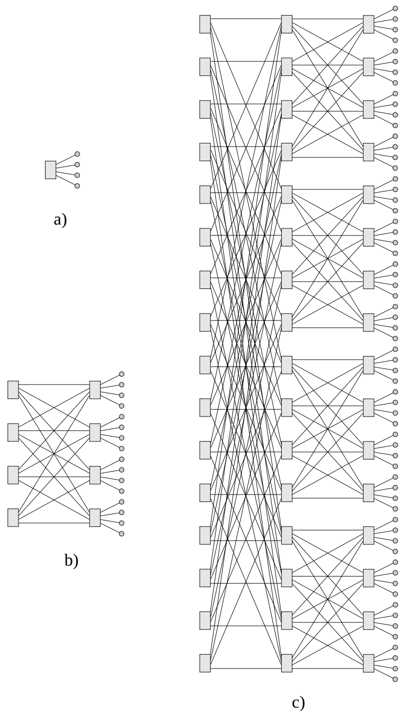


**Figure 2. $4$-ary $n$-trees of dimension 1, 2 and 3**

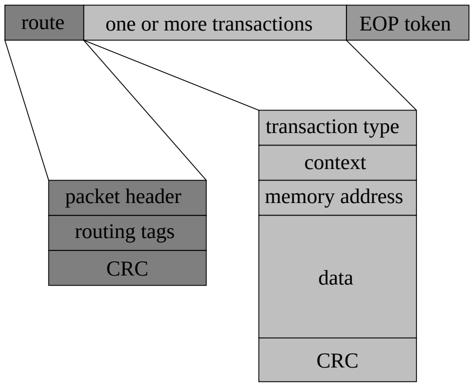


**Figure 3. Packet Transaction Format**

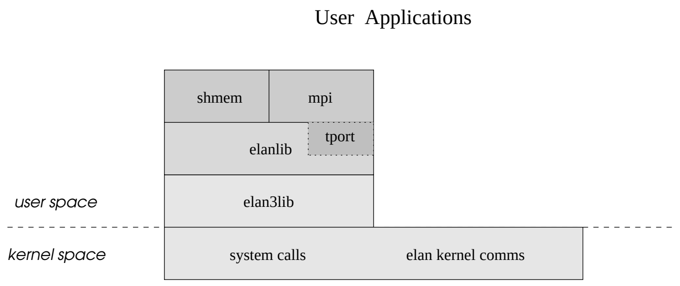


**Figure 4. Elan3 programming library hierarchy**

The transmission of each packet is pipelined into the network using wormhole flow control. At link level, each packet is partitioned in smaller units called flits (flow control digits) [4] of 16 bits. The header flit opens a circuit between source and destination, and this path stays in place until the destination sends an acknowledgment to the source. At this point, the circuit is closed by sending and End Of Packet (EOP) token. It is worth noting that both acknowledgment and EOP can be tagged to communicate control information. So, for example, the destination can notify the successful completion of a remote non-write block transaction without explicitly sending an extra packet.

Minimal routing between any pair nodes can be accomplished by sending the message to one of the nearest common ancestors and from there to the destination. That is, each packet experiences two routing phases, an adaptive ascending phase to get to a nearest common ancestor, followed by a deterministic descending phase. The Elite switches can adaptively route a packet picking the least loaded link.

## 3 Programming libraries

The Elan network interface can be programmed using several programming libraries [19], as outlined in Figure 4. These libraries trade speed with machine independence and programmability. Starting from the bottom, Elan3lib is the lowest programming level available in user space which allows the access to the low level features of the Elan3. At this level, processes in a parallel job can communicate with each other through an abstraction of distributed virtual shared memory. Each process in a parallel job is allocated a virtual process id (VPID) and can map a portion of its address space into the Elan. These address spaces, taken in combination, constitute a distributed virtual shared memory. Remote memory (i.e., memory on another processing node) can be addressed by a combination of a VPID and a virtual address. Since the Elan has its own MMU, a process can select which part of its address space should be visible across the network, determine specific access rights (e.g. write- or read-only) and select the set of potential communication partners.

Elanlib is a higher level layer that frees the programmer from the revision-dependent details of the Elan, and extends Elan3lib with point-to-point, tagged message passing primitives (called Tagged Message Ports or Tports) and support for collective communication. Standard communication libraries as such MPI-2 [6] or Cray Shmem are implemented on top of Elanlib.

### 3.1 Elan3lib

The Elan3lib library supports a programming environment where groups of cooperating processes can transfer data directly, while protecting process groups from each other in hardware. The communication takes place at user level, with no data copying, bypassing the operating system. The main features of Elan3lib are: (1) event notification, (2) the memory mapping and allocation scheme and (3) remote DMA transfers.

#### 3.1.1 Event Notification

Events provide a general purpose mechanism for processes to synchronize their actions. The mechanism can be used by threads running on the Elan and processes running on the main processor. Events can be accessed both locally and remotely. Thus, processes can be synchronized across the network, and events can be used to indicate the end of a communication operation, such as a completion of a remote DMA. Events are stored in Elan memory, in order to guarantee the atomic execution of the synchronization primitives[4]. Processes can wait for an event to be triggered by blocking or polling. In addition, an event can be

[4]The current PCI bus implementations cannot guarantee atomic execution, so it is not possible to store events in main memory.

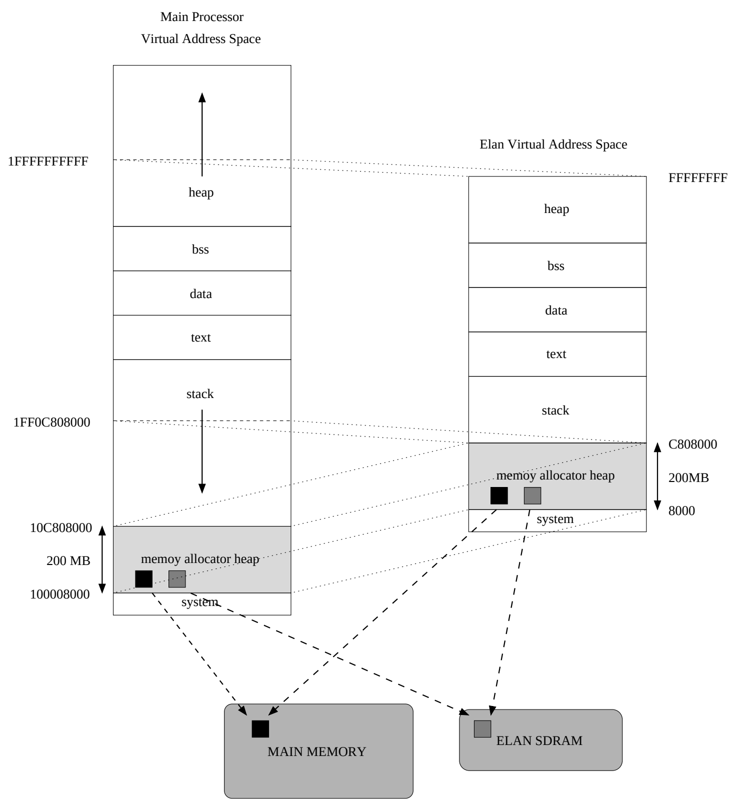


**Figure 5. Virtual Address Translation**

tagged as being a block copy event. The block copy mechanism works as follows. A block of data in Elan memory is initialized to hold a pre-defined value. An equivalent sized block is located in main memory, and both are in the user's virtual address space. When the specified event is set, for example when a DMA transfer has completed, a block copy takes place. That is, the block in Elan memory is copied to the block in main memory. The user process polls the block in main memory to check its value, (for example, bringing a copy of the corresponding memory block into the L2 cache) without having to poll for this information across the PCI bus. When the value is the same as that initialized in the source block, the process knows that the specified event has happened.

### 3.1.2 Memory Mapping and Allocation

The MMU in the Elan can translate between virtual addresses written in the format of the main processor (for example, a 64-bit word, big Endian architecture as the AlphaServer) and virtual addresses written in the Elan format (a 32-bit word, little Endian architecture). For a processor with a 32-bit architecture (for example an Intel Pentium), a one-to-one mapping is all that is required.

In Figure 5 the mapping for a 64-bit processor is shown. The 64-bit addresses starting at 0x1FF0C808000 are mapped to Elan's 32 bit addresses starting at 0xC808000. This means that virtual addresses in the range 0x1FF0C808000 to 0x1FFFFFFFFFF can be accessed directly by the main processor while the Elan can access the same memory by using addresses in the range 0xC808000 to 0xFFFFFFFF. In our example, the user may allocate main memory using malloc and the process heap may grow outside the region directly accessible by the Elan delimited by 0x1FFFFFFFFFF. In order to avoid this problem, both main and Elan memory can be allocated using a consistent memory allocation mechanism. As shown in Figure 5 the MMU tables can be set up to map a common region of virtual memory called *memory allocator heap*. The allocator maps physical pages, of either main or Elan memory into this virtual address range on demand. Thus, using allocation functions provided by the Elan library, portions of virtual memory (1) can be allocated either from main or Elan memory, and (2) the MMUs of both main processor and Elan can be kept consistent.

For efficiency reasons, some objects can be located on the Elan, for example communication buffers or DMA descriptors which the Elan can process independently of the main processor.

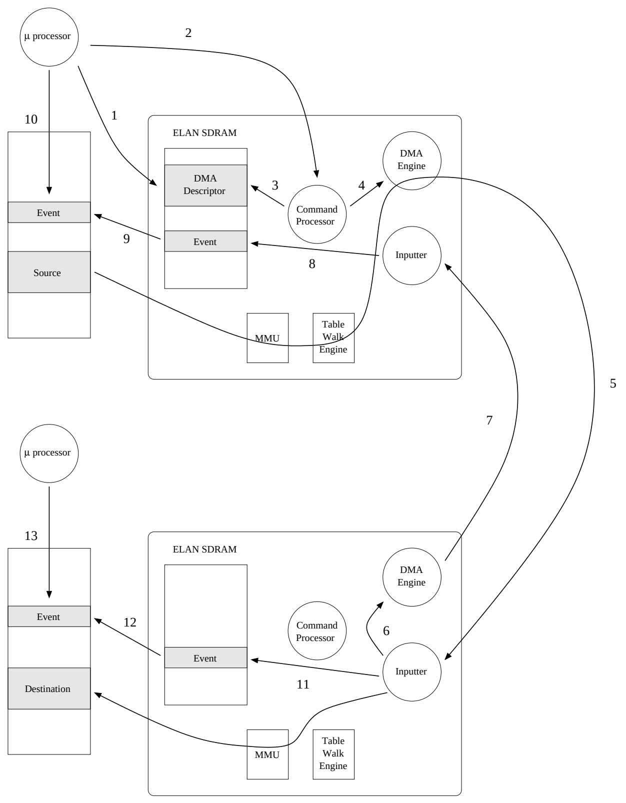


**Figure 6. Execution of a Remote DMA. The sending process (1) initializes the DMA descriptor in the Elan memory and (2) communicates the address of the DMA descriptor to the command processor. The command processor (3) checks the correctness of the DMA descriptor and (4) adds it to the DMA queue. The DMA engine (5) performs the remote DMA transaction. Upon completion the remote inputter (6) notifies the DMA engine which (7) sends an ack to the source Elan. Source (8-10) and destination (11-13) events can be notified, if needed**

#### 3.1.3 Remote DMA

The Elan supports remote DMA (Direct Memory Access) transfers across the network, without any copying, buffering or operating system intervention. The process that initiates the DMA fills out a DMA descriptor, which is typically allocated on the Elan memory for efficiency reasons. The DMA descriptor contains the VPIDs of both source and destination, the amount of data, the source and destination addresses, two event locations (one for the source and the other for the destination process) and other information used to enhance fault tolerance. The typical steps of remote DMA are outlined in Figure 6.

### 3.2 Elanlib and Tports

Elanlib is a machine independent library that integrates the main features of Elan3lib with the Tports. Tports provide basic mechanisms for point-to-point message passing. Senders can label each message with a tag, the sender identity and the size of the message. This is known as the *envelope*. Receivers can receive their messages selectively, filtering them according to the identity of the sender and/or a tag on the envelope. The Tport layer handles communication via shared memory for processes on the same node. It is worth noting that the Tports programming interface is very similar to MPI [23].

Elanlib provides support for collective communication operations (those that involve a group of processes). The most important collective communication primitives implemented in Elanlib are: (1) the barrier synchronization and (2) the broadcast.

## 4 Experimental Framework

We tested the main features of the QsNET on a 64-node cluster of Compaq AlphaServer ES40s, running Tru64 Unix. Each AlphaServer node is equipped with 4 Alpha 667MHz 21264 processors, 8 GB of SDRAM and two 64-bit, 33MHz PCI I/O buses. The Elan3 QM-400 card is attached to one of these buses and links the SMP to a quaternary fat tree of dimension three, as the one shown in Figure 2 c).

Unless otherwise stated, the communication buffers are allocated in Elan memory in order to isolate I/O bus-related performance limitations, except for the ping tests, whose goal is to provide basic performance results that are a reference point for the following experiments.

### 4.1 Unidirectional Ping

We analyze the latency and bandwidth of the network by sending messages of increasing sizes. In order to identify different bottlenecks, the communication buffers are placed either in main or in Elan memory, using the allocation mechanisms provided by Elan3lib described in Section 3.1.

At Elan3lib level the latency is measured as the elapsed time between the posting of the remote DMA request and the notification of the successful completion at the destination. The unidirectional ping tests for MPI are implemented using matching pairs of blocking sends and receives.

### 4.2 Bidirectional Ping

The unidirectional ping experiments can be considered as the "peak performance" of the network. By sending packets in both directions along the same network path we can expose several types of bottlenecks.

For example, the Elan microcode interleaves four activities, DMA engine, inputter, command processor and thread processor. This test can evaluate how the DMA engine and the inputter can work with bidirectional traffic. Also the link-level flow control requires the transmission of control information, which can lead to a degradation of the unidirectional performance in the presence of bidirectional traffic. Bidirectional traffic is typically generated by permutation patterns in which a node is both source and destination.

### 4.3 Uniform Traffic

The uniform traffic is one the most frequently used traffic patterns for evaluating the network performance. With this pattern each node randomly selects its destinations. This distribution provides what is likely to be an upper bound on the mean internode distance because most computations exhibit some degree of communication locality [9][5].

### 4.4 Permutation Patterns

These experiments provide information regarding the performance of the network under communication patterns that take into account the behavior of parallel numerical algorithms usually employed by scientific applications. In these patterns, each node selects a fixed destination for all its transactions.

The permutation patterns tested were:

- Bit-reversal. The node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with node $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$.
- Butterfly. The node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with node $a_0, a_{n-2}, \ldots, a_1, a_{n-1}$ (swap the most and least significant bits).
- Complement. The node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with node $\overline{a_{n-1}, a_{n-2}, \ldots, a_1, a_0}$.
- Matrix transpose. The node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with node $a_{\frac{n}{2}-1}, \ldots, a_0, a_{n-1}, \ldots, a_{\frac{n}{2}}$.
- Perfect-shuffle. The node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with node $a_{n-2}, a_{n-3}, \ldots, a_0, a_{n-1}$ (rotate left 1 bit).

- Neighbor. Nodes are divided in pairs of adjacent nodes, for example nodes 0 and 1, 2 and 3, $n$ and $(n+1)$ with $n$ even number, and each node communicates with its buddy. This pattern theoretically provides the best performance achievable because all the traffic is routed through the first level of switches, therefore avoiding network contention.

It is worth noting that all these communication patterns can be routed off-line without conflicts by a fat-tree. So, all the performance degradation is introduced by the routing and flow control algorithms [12].

### 4.5 Hot-spot

Under hot-spot traffic, a set of communication partners try to read from or write into the same memory block. This experiment models the behavior of a single I/O server being accessed by multiple clients and provides basic results for better understanding the network. This localized communication pattern can lead to a severe form of congestion known as *tree saturation* [18], which can seriously degrade the overall performance of the interconnect.

### 4.6 Multiple Hot-spots

The network traffic generated by a parallel job that is performing input/output can be modeled with a collection of hot-spots, where each hot-spot is a node that acts as an I/O server[5] and is the target of multiple messages originated by the other nodes. This test has been designed to analyze the behavior of the network when one parallel job is performing transactions over several hot nodes. We address five distinct performance dimensions.

1. I/O read/write ratio: the ratio between I/O reads and writes is a number between 0 and 1, with 0 being all reads and 1 all writes.
2. The inter-arrival time between two I/O messages issued by a client node can be either uniformly or exponentially distributed.
3. I/O traffic: this parameter defines the access pattern to the I/O nodes. Two patterns have been analyzed:
   (a) random I/O, each node performing I/O randomly selects its destination for every transaction and
   (b) deterministic I/O, each node uses a fixed destination for all its transactions.
4. I/O node mapping (hot-node mapping): this parameter defines the placement of the I/O nodes in the cluster. Two alternatives have been tested:
   (a) clustered, I/O nodes located in consecutive nodes at the higher nodes locations and
   (b) distributed, I/O nodes uniformly distributed through the cluster.
5. Application mapping: defines whether the application runs on the I/O nodes (shared I/O) or not (dedicated I/O).

Figure 7 describes the types of I/O and application mappings used in the experiments with a configuration of 64 nodes. In the clustered mapping, 8 I/O nodes are placed in the upper part of the network (the dark ones), while the remaining 56 nodes are dedicated to computation. For this type of I/O mapping, we consider two types of application mappings. In the "Shared I/O Mapping" all 64 nodes generate I/O traffic. In this case, the I/O nodes are both source and destination of the I/O traffic. In the "Dedicated I/O Mapping", only the first 56 nodes inject I/O traffic into the network. The I/O nodes are only sinks of the I/O traffic, as outlined by the arrow below the first row.

With distributed I/O mapping, shown in the second row of Figure 7, the I/O nodes are scattered with a stride of 8 over all the nodes. In this case we have an I/O node every 8 nodes. As in the clustered approach, we distinguish the two cases where the I/O nodes do and do not inject messages into the network.

[5]For this reason in the rest of the paper multiple hot-spots and I/O traffic are used interchangeably.

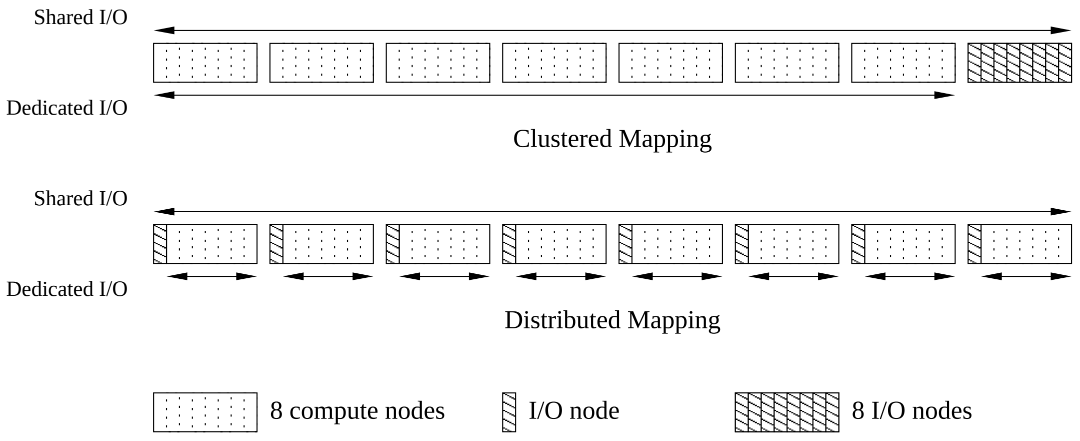


**Figure 7. I/O and Application Mappings with 64 nodes and 8 I/O nodes. The arrows highlight the nodes that inject I/O traffic into the network.**

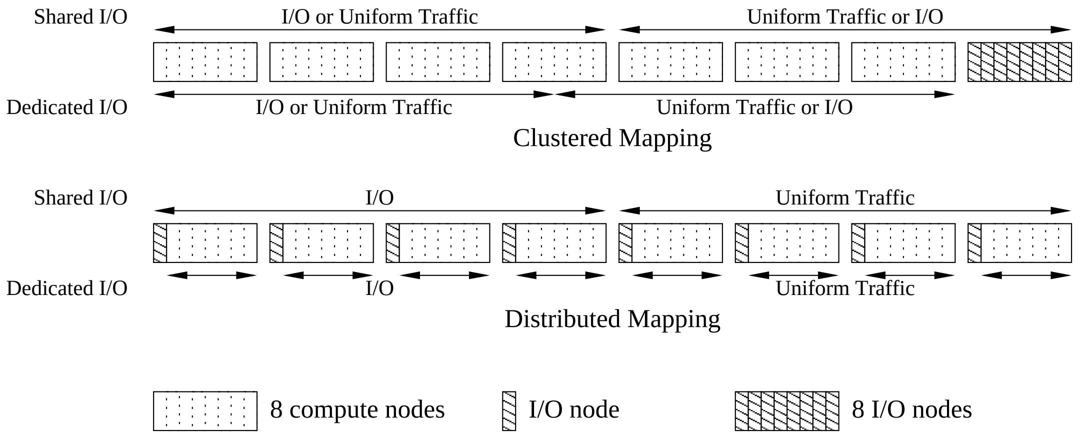


**Figure 8. I/O and Application Mappings with 64 nodes and 8 I/O nodes with combined traffic.**

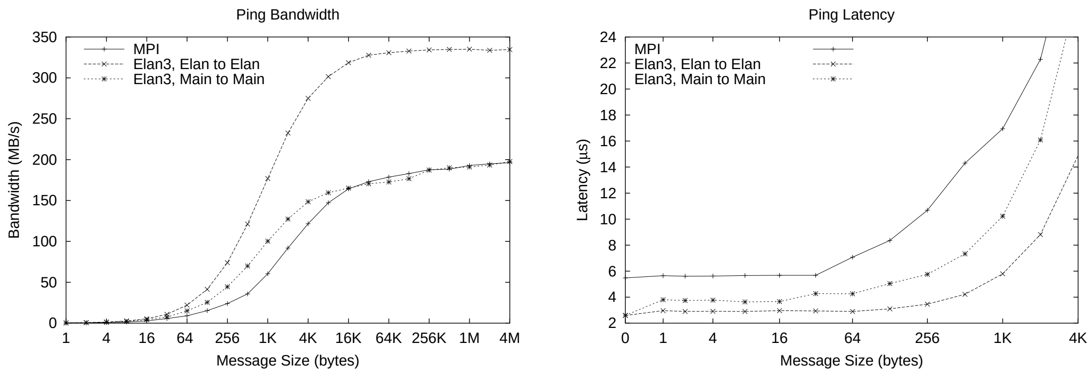
## 4.7 Combined Traffic

With this benchmark we study how a parallel job that is executing I/O traffic can affect the communication performance of another parallel job that is running concurrently. We perform the tests by running two parallel jobs in the cluster, each one using half of the available nodes. The I/O job generates traffic as described in Section 4.6, while the compute job injects uniform traffic. We analyze the Cartesian product of several performance dimensions, which are outlined in Figure 8. In particular,

1. I/O node mapping: we consider clustered (the upper row of Figure 8) and distributed mapping (the lower row of Figure 8). When we have the clustered I/O the position of both applications in the cluster may have implications on performance. For this reason, we distinguish two further cases. The compute job is mapped onto the lower half of the network and the job performing I/O on the higher half, and the symmetric case. As shown in the upper row of Figure 8, this determines the role (either I/O or compute) of the job mapped closer to the I/O nodes.

2. Application mapping: the two applications running in this test use half of the available nodes with two different approaches. If "Shared I/O Mapping" is used each application is mapped to 32 nodes (even if it overlaps with I/O servers). On the other hand, with "Dedicated I/O Mapping" the I/O nodes are dedicated servers and each application is mapped to 28 nodes.

3. I/O load: the I/O job can inject messages into the network with increasing load.



(a) (b)

**Figure 9. Unidirectional Ping**

## 5 Experimental Results

### 5.1 Unidirectional Ping

Figure 9 a) shows the performance of the unidirectional ping. The peak bandwidth of 335 MB/s is reached when both source and destination buffers are placed in the Elan memory. The maximum amount of data payload that can be sent by the current Elan implementation in a packet is 320 bytes, partitioned in five low-level write-block transactions of 64 bytes. For this packet format, the overhead is 58 bytes, for the message header, CRCs, routing info, etc. This implies that the delivered peak bandwidth is approximately 396 MB/s, or 99% of the nominal bandwidth (400 MB/s).

But the asymptotic bandwidth for main memory to main memory communication is only 200 MB/s for both Elan3lib and MPI. These results show that the PCI interface running at 33MHz is the bottleneck for this type of communication.

Figure 9 b) shows the latency in the range $[0 \ldots 4KB]$. With Elan3lib the basic latency for 0-byte messages is only $2.2$ $\mu$s and is almost constant at $2.4$ $\mu$s for messages up to $64$ bytes. We note an increase in the latency at MPI level, compared to the latency at the Elan3lib level, from approximately $2$ $\mu$s to $5.5$ $\mu$s. While at Elan3lib level the latency is mostly hardware, MPI needs to run a thread in the Elan microprocessor in order to match the message tags: this introduces the extra overhead responsible for the higher latency.

### 5.2 Bidirectional Ping

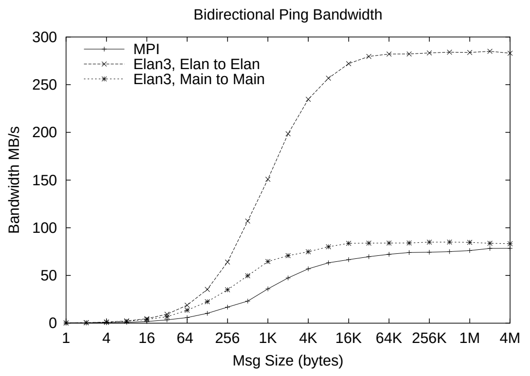
Figure 10 a) shows how the bidirectional bandwidth is degraded by the PCI bus. When the communication buffers are in Elan memory, the asymptotic bandwidth is 280 MB/sec, a slight performance degradation from the 335 MB/sec of the unidirectional ping. With main memory to main memory communication the bandwidth drops to 80 MB/sec, a performance degradation caused by the PCI bus of the AlphaServer[6], that is not able to efficiently interleave the bidirectional traffic. Given this substantial limitation, in the following experiments we will place the communication buffers in Elan memory, in order to inject messages into the network at full speed.

### 5.3 Uniform Traffic

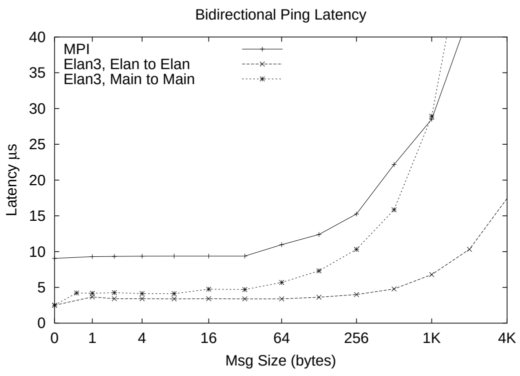
The results obtained for the uniform traffic pattern with 256 KB messages are shown in Figures 11 a), b) and c) for 16, 32 and 64 nodes configurations. The maximum sustained bandwidth versus the network size is depicted in Figure 11 d).

In the experiments we consider uniform and exponential distributions for both message size (identified by the tag S in the figures) and inter-arrival time (tag T). The results show that small network configurations are slightly sensitive to these distributions, providing better results when both message size and inter-arrival times are uniformly distributed. This performance gap narrows with larger configurations and almost disappears with 64 nodes.

[6]Other PCI buses running at 66 Mhz rather than at 33 Mhz, for example those based on the Serverworks HE chipsets, don't suffer from these limitations and can provide almost the same communication performance when the buffers are placed in main or in Elan memory [15].



(a)



(b)

**Figure 10. Bidirectional Ping**

In Figure 11 d) we can see that the network performance doesn't scale with the network size. For example, with 64 nodes the asymptotic bandwidth is less than 120 MB/sec, only 42% of the maximum bidirectional bandwidth, which is an upper bound for uniform traffic. This performance degradation is caused by the flow control algorithms. Each packet keeps an open circuit between source and destination until the packet is successfully delivered, which increases the probability of having other packets blocked waiting for an available path. This problem is partially alleviated by the two virtual channels, that offer an escape path when one of the channels is busy, but they are not enough to guarantee scalability over a large number of nodes.

The results for the maximum bandwidth versus message size are depicted in Figure 12, and show that there is a performance penalty when the average message size is larger than 16KB.

## 5.4 Permutation Patterns

Figure 13 shows the accepted bandwidth versus the offered bandwidth for all the permutation patterns generated on a network configuration with 64 nodes. Average results for 256 KB messages are shown for uniform and exponential time and message size distributions. In Figure 14 the same permutation patterns are analyzed as a function of the message size. All these patterns share two important features.

1. The nodes either generate bidirectional traffic, that is each node is both a communication source and sink, or do not engage in any network communication. In some traffic patterns some nodes do not generate any network traffic, for example those with a palindrome bit string as node id in the bit-reversal traffic, because both source and destination are placed on the same node. The optimal performance for each communicating node is thus limited by the asymptotic bidirectional bandwidth.

2. All the traffic patterns can be routed by a fat tree in a congestion-free manner by an off-line routing algorithm, so the performance degradation is caused by the on-line routing and flow control algorithms.

Taking into account the results obtained with the bidirectional ping tests (Section 5.2 and Figure 10 a), Elan to Elan communication, with asymptotic bandwidth of 280 MB/sec), we have three permutation patterns (neighbor, butterfly and complement) that achieve optimal performance and three other patterns (shuffle, transpose and bit-reversal) that cannot be routed efficiently by the network.

Although neighbor and complement permutations get the same performance, they generate traffic patterns with widely different topological characteristics. In fact, with the neighbor traffic every destination is two hops away, with each pair of communicating partners connected directly to the same Elite switch. This pattern does not generate any traffic that goes above the first level of switches. On the other hand, with complement traffic, every message needs to reach one of the top-level switches and must be routed along one of the longest paths (6 hops, in our network configuration). Another interesting property of complement traffic is that it can potentially keep all the communication links of the network active at the same time.

The asymptotic performance of shuffle, transpose and bit-reversal ranges between 165 and 200 MB/sec. These traffic patterns cannot be handled efficiently by the routing and flow control algorithms. In order to provide more insight into the

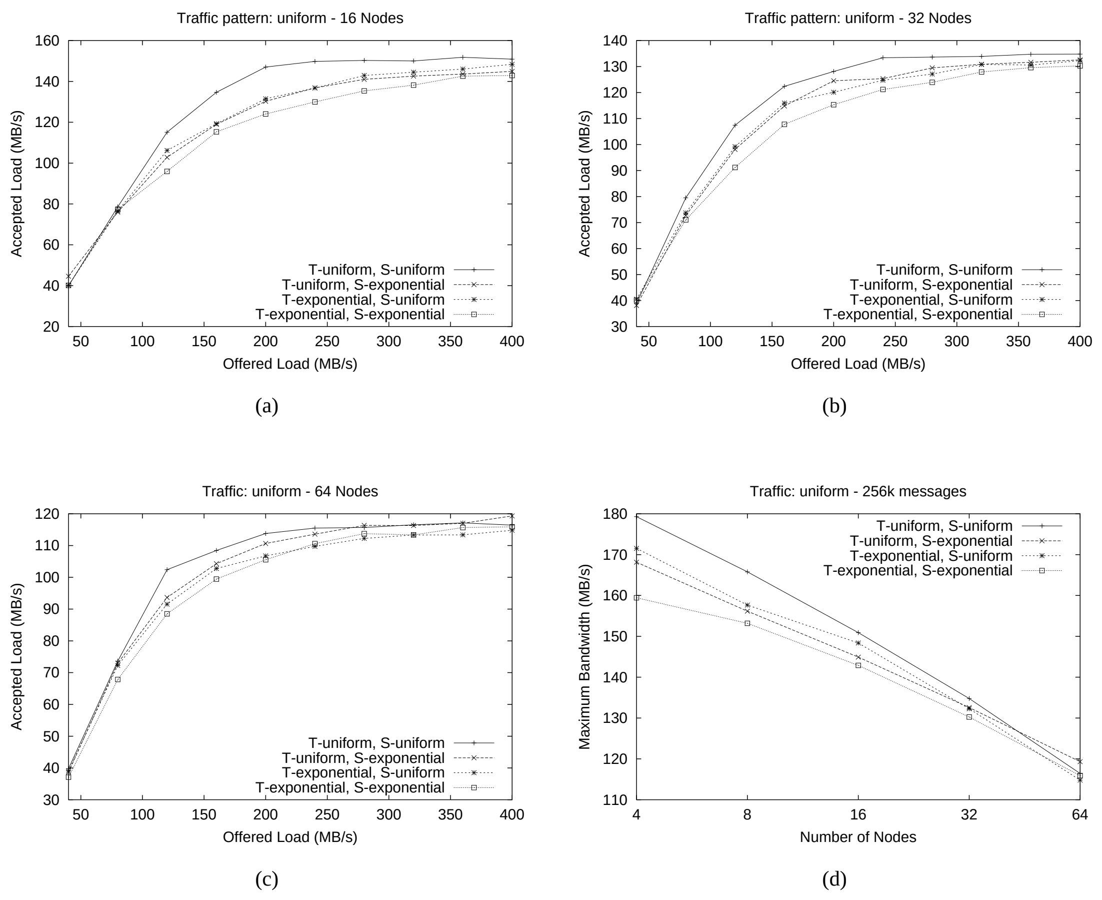


**Figure 11. Uniform Traffic Scalability**

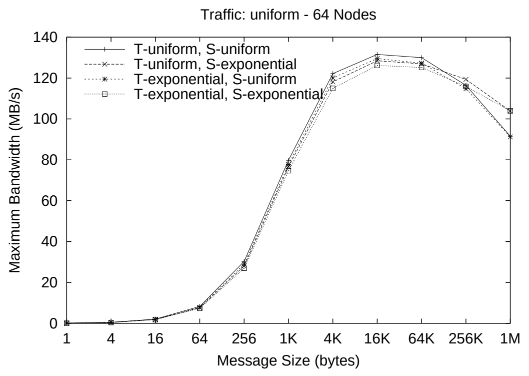


**Figure 12. Uniform Traffic Maximum Bandwidth**

network behavior, we will use a graphic representation of the network congestion. In Figure 15 we can see two congestion matrices, for the bit-reversal and complement traffic. The goal of a congestion matrix is to identify the hot switches and links, where congestion may arise. The matrix is composed of 3 rows with 16 switches each, and represents a quaternary fat tree with 64 nodes. Each block has 4 numbers in the upper side representing the load on the links connecting the switch to the upper level switches, and 4 numbers in the lower side representing the load on the links connecting to the lower level switches or to the processing nodes. These numbers represent the sum of the delay experienced by the packets incoming from the corresponding link, normalized with the value of the most loaded link. The number in the center represents the sum of the wait times over all links in the switch, normalized with the value of the most loaded switch. A darker background highlights the congested links or switches.

The congestion matrix of the complement traffic ( Figure 15 a)) shows that all the delays are evenly distributed for the switches of the same level. The packets experience an increasing delay as they move forward to their destination, from 17 units when they enter the first level switch, to 37 units when they enter the second switch, and so on to the most congested links from the nodes to the first level of switches on the way back. From the congestion matrix we can also gather that, for this pattern and network configuration, the top level of switches could be halved, without having a detrimental effect on the overall performance. As shown in [7], fat-trees can efficiently handle this type of communication pattern.

The congestion matrix of the bit-reversal traffic, shown in Figure 15 b), which is one of the patterns that cannot be efficiently routed by the network, displays several hot links connected to the first level of switches. The congestion created by these hot-spots is the reason for an uneven distribution of bandwidth among destinations and the observed performance degradation.

## 5.5 Hot-spot

In this experiment we attempt to write into the same memory location on node 0 from an increasing number of processors (one per SMP). This test provides information on the behavior of a single I/O node when serving multiple simultaneous requests. Previous results [15] have shown that read and write operations provide no significant differences. The aggregate bandwidth plots are depicted in Figure 16 a) for 1 MByte messages, using uniform and exponential time (T tag) and message size (S tag) distributions.

The curves are approximately flat up to 32 nodes, reaching 337 MB/s, while an 8% decrease is observed for 64 nodes. This is due to the extra contention experienced in the third level of switches (Figure 2 c)) and to the longer delays needed to release a circuit when there are more than 40 communication partners. Figure 16 b) shows the distribution of the delivered bandwidth per node on a 64-node configuration, and provides more insight into this problem. It can be seen that the nodes are distributed in three bandwidth groups: nodes from 0 to 3 get approximately 8 MB/s, nodes 4 to 15 get approximately 4.8 MB/s and nodes 16 to 63 get around 4.3 MB/s. This uneven distribution of bandwidth is due to the various areas of contention that a given packet can traverse. Considering that 0 is the hot node, packets sent from nodes 1 to 3 have a single contention point in the switch they are directly connected. Packets sent from nodes 4 to 15 have two potential contention points, one in the second level of switches and the other in the destination switch. Finally, packets sent from nodes 16 to 63 traverse three contention stages in the three levels of switches.

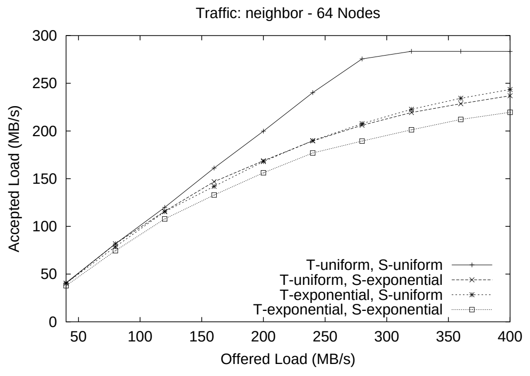
## 5.6 Multiple Hot-spots

This test is designed to analyze the behavior of the network when one parallel application is performing transactions over several I/O nodes, as described in Section 4.6.

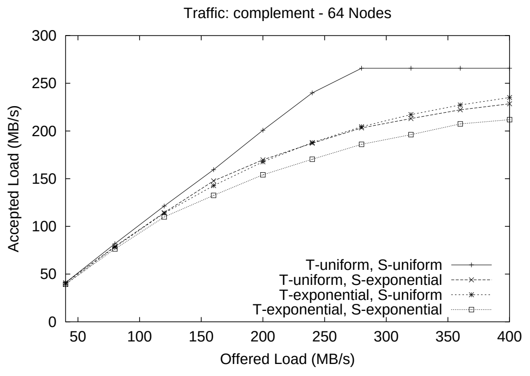
The experiments are performed on a 64-node configuration with 8 hot-spots, as shown in Figure 7. The average message size is 1 MB (exponentially distributed), with inter-arrival times uniformly and exponentially distributed. The results show very little sensitivity to the fraction of read/write requests, so we will omit the related experiments and we will report results for 0.5 read/write ratio. Figures 17 and 18 display the accepted load of the I/O nodes versus the offered load for random and deterministic traffic, respectively. In each figure there is a graph for the clustered I/O mapping (sub figure a)) and a graph for the distributed I/O mapping (sub figure b)). Each graph displays curves for the two application mappings presented in Section 4.6 (shared I/O and dedicated I/O). The asymptotic bandwidths are summarized in Table 1 for the most significant performance dimensions, I/O traffic and I/O mapping.

The results show that a deterministic destination pattern (Figure 18) always provides better performance than a random selection of destinations (Figure 17).

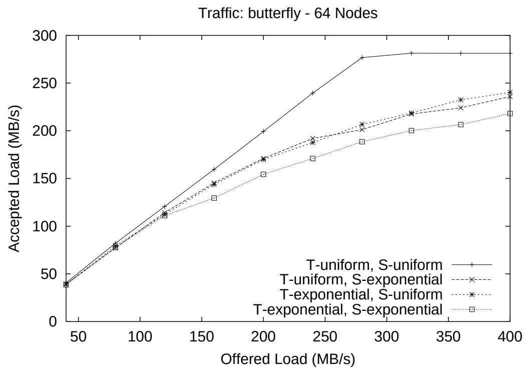
The I/O mapping has a significant effect on performance too. Better results are always obtained with distributed I/O (Figures 17 b) and 18 b)). This is due to the fact that the distribution of I/O nodes through the cluster evenly spreads the traffic across the network, while with the clustered mapping we generate a large hot-spot on one side of the network. It is worth noting that



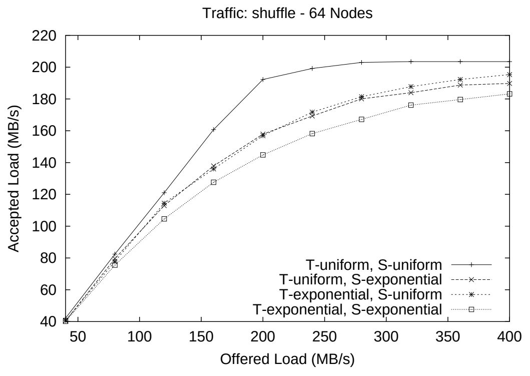
(a)



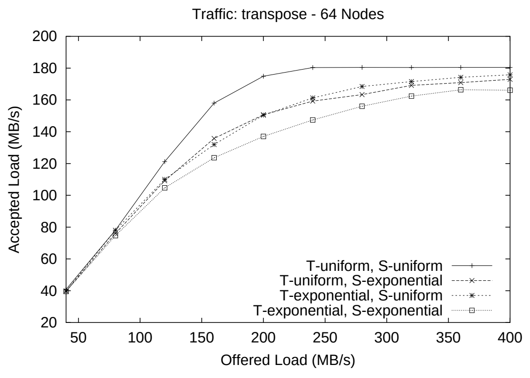
(b)



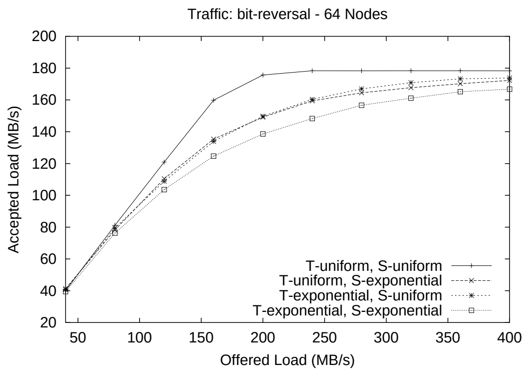
(c)



(d)

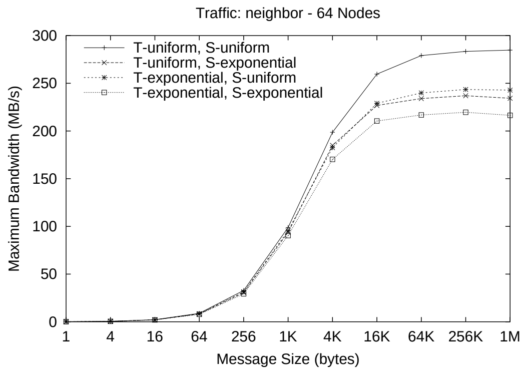


(e)

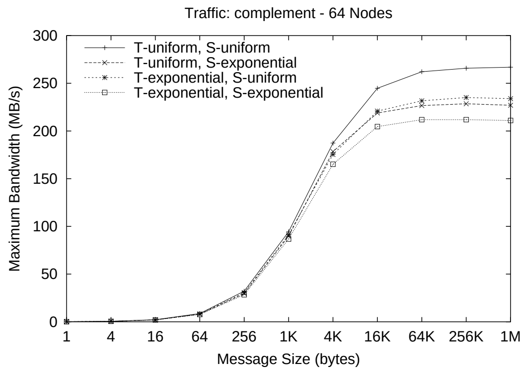


(f)

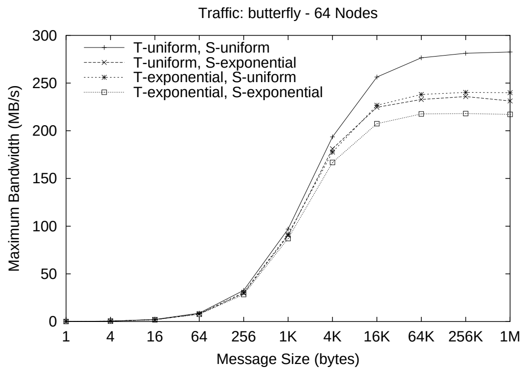
**Figure 13. Permutation Patterns Accepted Load**



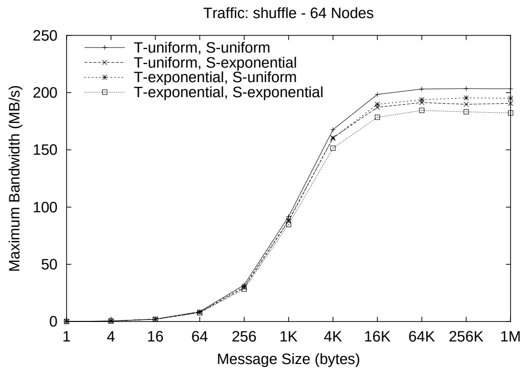
(a)



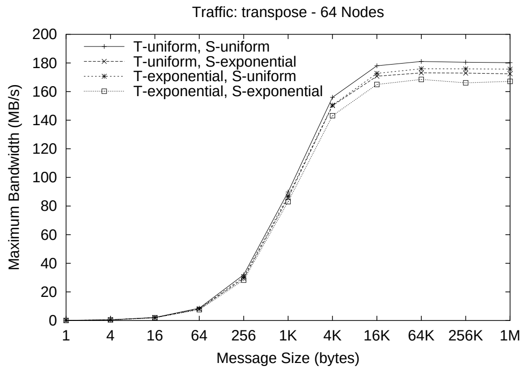
(b)



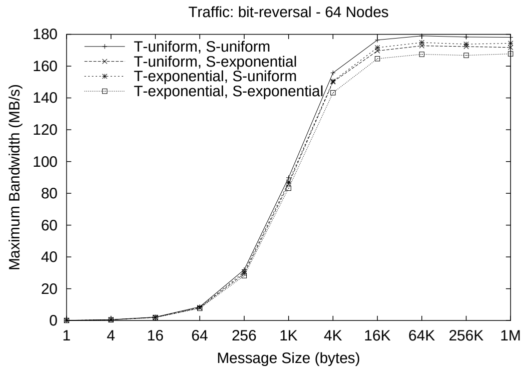
(c)



(d)



(e)



(f)

**Figure 14. Permutation Patterns Maximum Bandwidth**

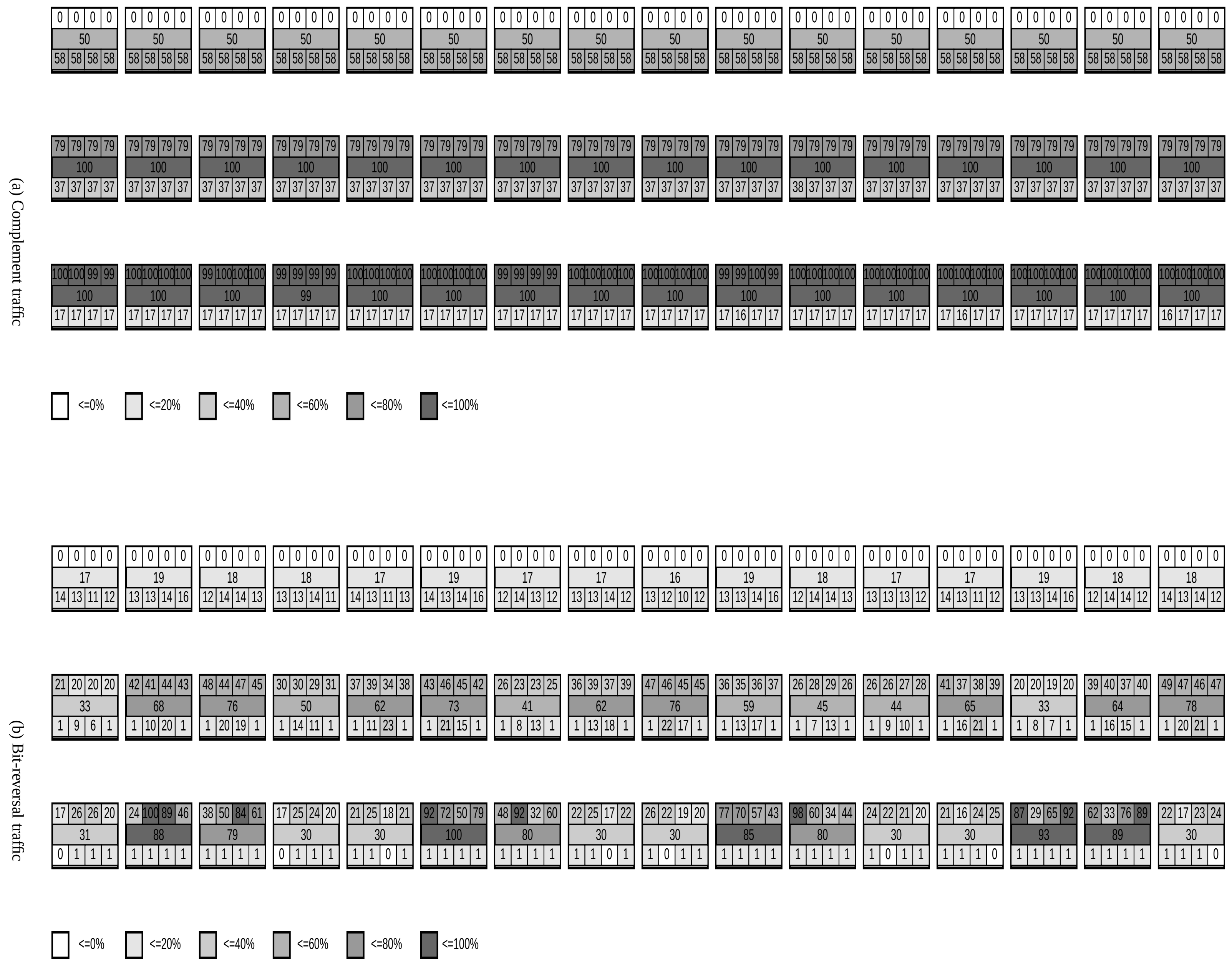
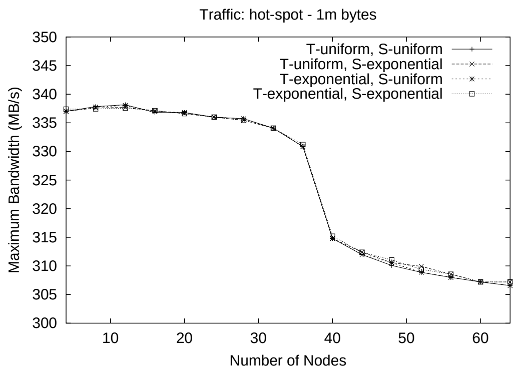
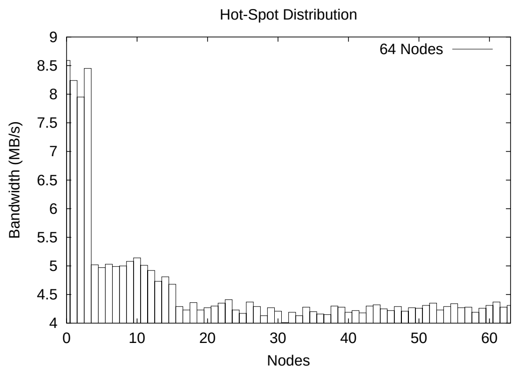


**Figure 15. Congestion Matrices**



(a)



(b)

**Figure 16. Write Hot-spot**

| | Clustered I/O | Distributed I/O |
|---|---|---|
| Random Traffic | 196 MB/s | 234 MB/s |
| Deterministic Traffic | 320 MB/s | 338 MB/s |

**Table 1. Multiple Hot-spots Maximum Accepted Load Summary**

with distributed mapping each I/O node is connected to a distinct switch, while with clustered I/O four I/O nodes share the same switch. Thus, the adjacent allocation of I/O nodes worsens the contention in the network.

The application mapping has no significant effect when using distributed I/O (Figures 17 b) and 18 b)) and a small effect with clustered I/O (Figures 17 a) and 18 a)). In the latter case, the I/O nodes deliver slightly higher asymptotic bandwidth when shared I/O is used (between 15 and 20 MB/s), either with deterministic or random traffic.

## 5.7 Combined Traffic

With this benchmark we study how a parallel job that is executing I/O traffic can affect the communication performance of another parallel job that is running concurrently.

The I/O job generates random traffic, shown to produce high network contention (Section 5.6), with exponential distribution of the inter-arrival times and messages of 1 MB. We consider three I/O loads, with increasing intensity, which are expressed as fraction of the asymptotic load that can be injected into the network by a node (0.1, 0.3 and 0.5). The other parallel job uses uniform traffic and 256KB messages.

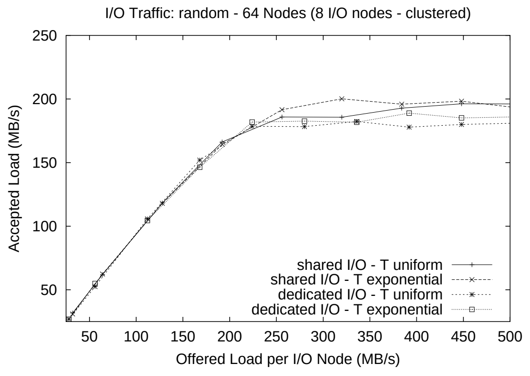
With clustered I/O, we distinguish two cases: the compute job allocated on the first half of the machine, and the symmetric case where the I/O job is allocated there (Section 4.7). In the graphs we use the label *1c*[7] to indicate the first case and *1i* to indicate the second case.

The graphs in Figure 19 show the accepted bandwidth of the compute job. Considering the type of traffic (uniform) and the message size, the optimal asymptotic bandwidth of the compute job is about 130 MB/s. Any performance degradation indicates that the background I/O traffic interferes with the compute job. Figures 19 a), c) and e) consider the shared I/O mapping. In Figures 19 b), d) and f) we can see the results for the dedicated I/O. We can clearly see that :

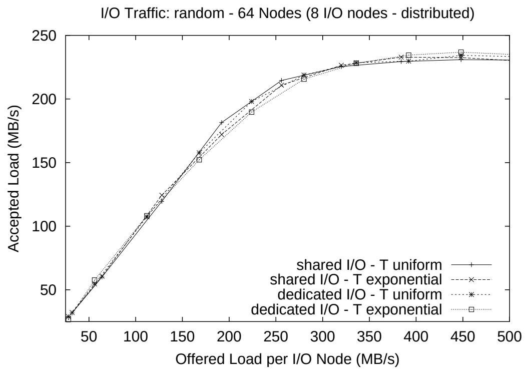
1. When the jobs do not run on the I/O nodes (the application mapping is dedicated I/O) there is no interference. This is a very powerful result that shows that any job mapping and I/O mapping will perform well, as long as the processes of both jobs do not run on the I/O nodes.

2. This basic result is extended to another case. When the compute job is not mapped onto the I/O nodes (clustered 1c, the I/O job overlaps the I/O nodes), there is no interference.

[7] A shortcut to indicate that the first half of the machine is devoted to computation (the second half will be allocated to I/O in this case), with the I/O nodes clustered in the last segment of the network.

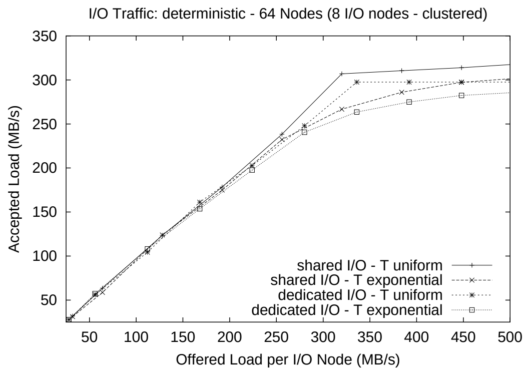


(a)

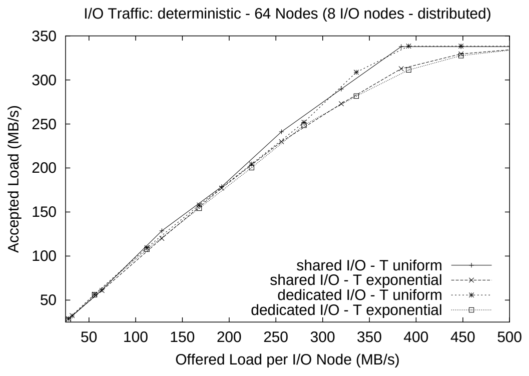


(b)

**Figure 17. Multiple Hot-spots with Random I/O Traffic**



(a)



(b)

**Figure 18. Multiple Hot-spots with Deterministic I/O Traffic**

3. When a fraction of the compute job is mapped onto the I/O nodes (clustered 1i, the compute job overlaps the I/O nodes) there is a substantial performance degradation, up to 40%, with any I/O load.

4. With distributed mapping the performance is sensitive to the I/O load. The higher the background I/O load the lower the accepted bandwidth.

Figure 20 summarizes the analysis of the results and highlights the performance regions where the computational job is affected by the background I/O traffic.

## 6 Conclusions

In this paper, we present a comprehensive description and evaluation of the Quadrics interconnection network (QsNET), which can prove particularly useful for network designers, library developers and users of high performance parallel clusters.

In the initial part we describe in depth the building blocks that compose the QsNET, the Elan network interface and the Elite routing chip. Relevant details are the internal architecture and the functional units of the Elan, and the network topology, routing algorithms and flow control algorithms that characterize the Elite. We also describe the hardware communication protocol that is at the base of the distributed virtual shared memory implemented by the QsNET.

In the performance evaluation section, we analyze the communication performance and the scaling properties of the network using a broad set of communication patterns, on a 64-node AlphaServer cluster. We start with the unidirectional and bidirectional pings to analyze the basic network performance. Results show a remarkable latency of 2.2 $\mu sec$ and a bandwidth over 300 MB/sec. The results with uniform traffic show that the network doesn't scale well, due to internal congestion. For example, with 64 nodes, the delivered asymptotic bandwidth is only 42% of the optimal bandwidth.

On the other hand, the QsNET can perform well on some important collective communication patterns, such as the complement and butterfly traffic. In order to identify the bottlenecks for the patterns that are not handled so efficiently by the network, like shuffle, transpose and bit-reversal, we visualize the network hot-spots using a congestion matrix that displays the aggregate wait times during the execution of the communication pattern for each link and routing switch.

We presented an extensive performance evaluation of several types of I/O traffic. Although this analysis applies only to the topology investigated, it can prove particularly useful for system and network designers and users of high-performance parallel clusters.

We first considered a single parallel job performing I/O and modeled the I/O traffic with a single hot-spot, representing a single I/O server, and multiple hot-spots, representing groups of I/O servers. The experimental results provided insight on several important open problems. We have shown that it is more efficient to distribute the I/O servers rather than cluster them in a single segment of the network, with a bandwidth increase of about 20%. Also, the performance is insensitive to both the fraction of I/O reads and writes and to the mapping of the parallel job, whose processes can be run on the I/O nodes without any noticeable performance degradation.

We then analyze how a job performing I/O can affect the communication performance of another job. Multiple jobs can be run concurrently without interference, as long as these jobs are not mapped on the I/O nodes. This is a powerful result, that gives a high degree of freedom when mapping multiple jobs. On the other hand, the I/O job can interfere with a compute job when processes of the compute job are mapped on the I/O nodes.

## Acknowledgements

The authors would like to thank the Quadrics team, David Addison, Jon Beecroft, Robin Crook, Moray McLaren, David Hewson, Duncan Roweth and John Taylor, for their invaluable support.

## References

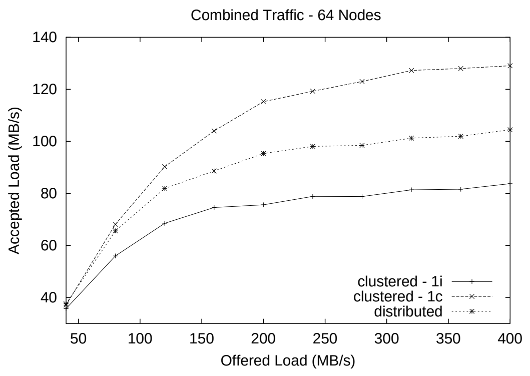
[1] *InfiniBand Specification 1.0a.* InfiniBand Trade Association, June 2001.

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawick, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, January 1995.

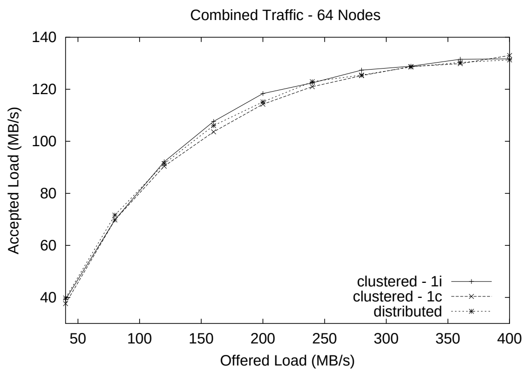
[3] Helen Chen and Pete Wyckoff. Simulation studies of Gigabit ethernet versus Myrinet using real application cores. In *Proceedings of CANPC'00, Workshop of High-Performance Computer Architecture*, Toulouse, France, January 2000.

[4] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

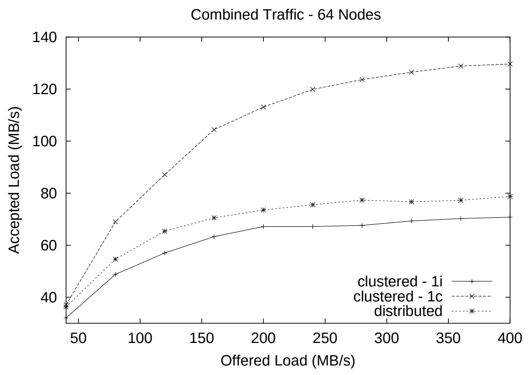
[5] José Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks: an Engineering Approach*. IEEE Computer Society Press, 1997.



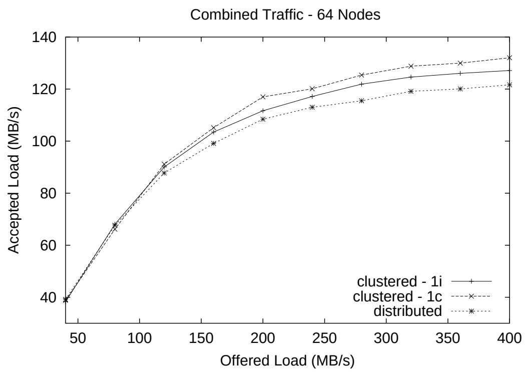
(a) Shared I/O with load 0.1



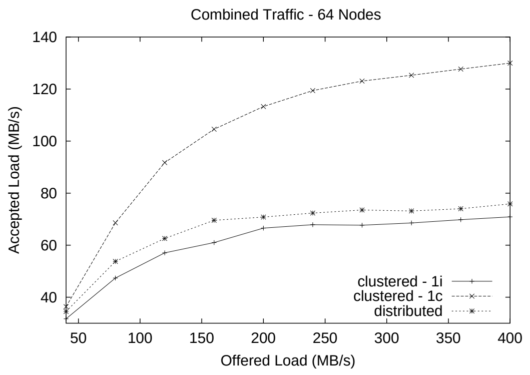
(b) Dedicated I/O with load 0.1



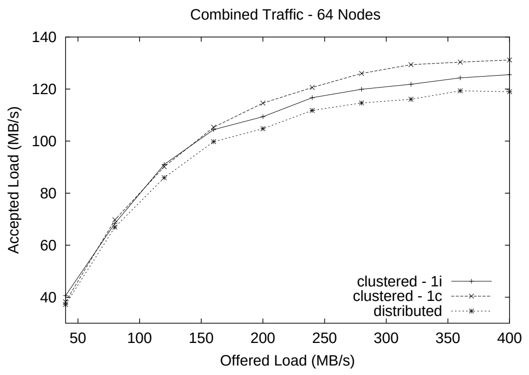
(c) Shared I/O with load 0.3



(d) Dedicated I/O with load 0.3



(e) Shared I/O with load 0.5



(f) Dedicated I/O with load 0.5

**Figure 19. Combined Traffic**

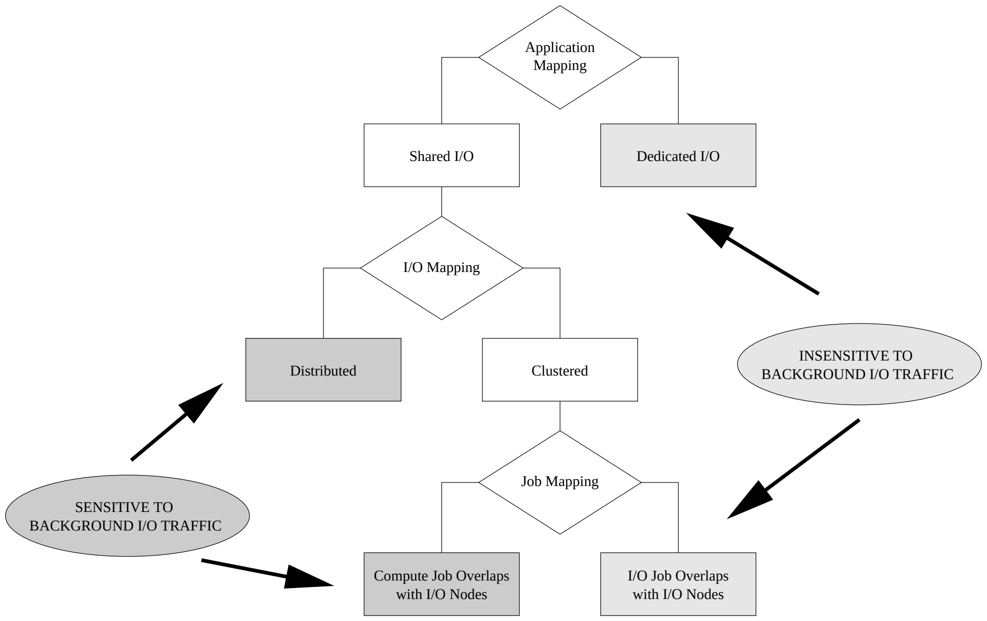

Application Mapping
Shared I/O
Dedicated I/O
I/O Mapping
Distributed
Clustered
INSENSITIVE TO BACKGROUND I/O TRAFFIC
Job Mapping
SENSITIVE TO BACKGROUND I/O TRAFFIC
Compute Job Overlaps with I/O Nodes
I/O Job Overlaps with I/O Nodes

**Figure 20. Combined Traffic Results Analysis**

[6] Al Geist, William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, William Saphir, Tony Skjellum, and Marc Snir. MPI-2: Extending the Message Passing Interface. In *Second International Euro-Par Conference, Volume I*, number 1123 in LNCS, pages 128–135, Lyon, France, August 1996.

[7] Steve Heller. Congestion-Free Routing on the CM-5 Data Router. In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 176–184, Seattle, Washington, USA, May 1994.

[8] Hermann Hellwagner. The SCI Standard and Applications of SCI. In Hermann Hellwagner and Alexander Reinfeld, editors, *SCI: Scalable Coherent Interface*, volume 1291 of *Lecture Notes in Computer Science*, pages 95–116. Springer-Verlag, 1999.

[9] Darren Kerbyson, Hank Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey Wasserman, and Mike Gittings. Predictive Performance and Scalability Modeling of a Large-Scale Application. In *Supercomputing 2001*, Denver, CO, November 2001.

[10] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[11] Lionel M. Ni, Yadong Gui, and Sherry Moore. Performance Evaluation of Switch-Based Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):462–474, May 1997.

[12] Fabrizio Petrini. *Communication Performance of Wormhole Interconnection Networks*. PhD thesis, Università degli Studi di Pisa, Dipartimento di Informatica, February 1997.

[13] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.

[14] Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg, and Adolfy Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network'. In *IEEE International Symposium on Network Computing and Applications 2001 (NCA 2001)*, Boston, MA, October 2001.

[15] Fabrizio Petrini, Adolfy Hoisie, Wu chun Feng, and Richard Graham. Performance Evaluation of the Quadrics Interconnection Network. In *Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.

[16] Fabrizio Petrini and Marco Vanneschi. $k$-ary $n$-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, pages 87–93, Geneva, Switzerland, April 1997.

[17] Fabrizio Petrini and Marco Vanneschi. Performance Analysis of Wormhole Routed $k$-ary $n$-trees. *International Journal on Foundations of Computer Science*, 9(2):157–177, June 1998.

[18] G. F. Pfister and V. A. Norton. Hot-spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

[19] Quadrics Supercomputers World Ltd. *Elan Programming Manual*, January 1999.

[20] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, January 1999.

[21] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, November 1999.

[22] Rich Seifert. *Gigabit Ethernet: Technology and Applications for High Speed LANs*. Addison-Wesley, May 1998.

[23] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI - The Complete Reference*, volume 1, The MPI Core. The MIT Press, 1998.

[24] D. Tolmie, T. M. Boorman, A. DuBois, D. DuBois, W. Feng, and I. Philp. From HiPPI-800 to HiPPI-6400: A Changing of the Guard and Gateway to the Future. In *Proceedings of the 6th International Conference on Parallel Interconnects (PI'99)*, October 1999.

[25] Werner Vogels, David Follett, Jenwi Hsieh, David Lifka, and David Stern. Tree-Saturation Control in the AC$^3$ Velocity Cluster. In *Hot Interconnects 8*, Stanford University, Palo Alto CA, August 2000.